Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 93 (2016) 982 – 987

Procedia Computer Science

6th International Conference On Advances In Computing & Communications, ICACC 2016, 6-8 September 2016, Cochin, India

Enhanced Merge Sort- a new approach to the merging process

Smita Paira[a], Sourabh Chandra[b,*], Sk Safikul Alam[b]

[a]*Dept of CSE,Calcutta Institute of Technology, Uluberia, Howrah-711316*

[b]*Dept of CSE, Calcutta Institute of Technology, Uluberia, Howrah-711316*

Abstract

One of the major fundamental issues of Computer Science is arrangement of elements in the database. The efficiency of the sorting algorithms is to optimize the importance of other sorting algorithms[11]. The optimality of these sorting algorithms is judged while calculating their time and space complexities[12]. The idea behind this paper is to modify the conventional Merge Sort Algorithm and to present a new method with reduced execution time. The newly proposed algorithm is faster than the conventional Merge Sort algorithm having a time complexity of $O(n \log_2 n)$. The proposed algorithm has been tested, implemented, compared and the experimental results are promising.

Peer-review under responsibility of the Organizing Committee of ICACC 2016

*Keywords:*Divide and Conquer; Internal Sorting; External Sorting; stack; Iterative; Recursive; Merge Sort

1. Introduction

One of the major and basic problems of Computer Science ever is the arrangement of elements in a given order. A number of Sorting algorithms provide solutions to these problems. Some of these algorithms are simple and spontaneous, like Insertion Sort while others are extremely complex, like Quick Sort yet provide quick results[4, 10].

Almost all sorting algorithms are problem specific[13]. Some algorithms can work on small data while others can work well on large data. Some are suitable for floating point values while others can work on lists having repeated values. We usually sort the lists in statistical order or in lexicographical order, in ascending or descending order[6, 7].

* Corresponding author. Tel.: +91-9830521934.
*E-mail address:*Sourabh.chandra@gmail.com

Peer-review under responsibility of the Organizing Committee of ICACC 2016
doi:10.1016/j.procs.2016.07.292

The conventional sorting algorithms can be divided into two classes based on the difficulty of the problems[1, 2, 3]. The algorithmic complexity is generally represented in Big- O notation of Time complexity where O represents the complexity of the algorithm and n represents the size of the list[5]. The two classes of algorithms are categorized as follows:-

- $O(n^2)$ or iterative algorithms- like Bubble Sort, Selection Sort, Insertion Sort and Shell Sort
- O(n log n) or recursive algorithms- like Heap Sort, Merge Sort and Quick Sort

The recursive or Divide and Conquer Sorting algorithms are slightly complex and more efficient than the iterative algorithms[8, 9]. Searching information from a list requires the list to be sorted in a sensible order.

2. Proposed algorithm

Among various Divide and Conquer sorting algorithms, Merge Sort has owned a wide range of applications. Such sorting algorithm has a time complexity of O(n log n) and can work better than the Insertion sort in case of large arrays[14]. It is even faster than the Quick Sort[10] if the list is in reverse order. In spite of having enough advantages, it becomes slow for small lists and consumes more stack space[5] with a complexity of O(n).

This paper deals with a new sorting approach that is based on the Divide and Conquer paradigm. It is a modified version of the conventional Merge Sort algorithm applying the Max Min algorithm[9] strategy. It also has a time complexity of O(n log n) but executes faster than the conventional part both in large and small lists. The general steps for the proposed algorithm is as follows:-

- Input the unsorted array
- Perform pair-wise sorting of each element of the array
- Divide the array into two sub-lists considering odd-even positions of the original array respectively
- Continue the above steps until the entire array is divided into sub-lists containing two elements each
- Merge and combine the sub-arrays
- Output the sorted array

The pseudo code, time complexity for the above algorithm and a brief comparison analysis with other conventional algorithms has been discussed in the upcoming sections.

3. Pseudo code for the proposed algorithm

```
Algorithm: Sort (L, first, last)
Where L: array of elements
     first: lower bound of L
      last: higher bound of L

Step 1: Initialise
        i.      n=last-first+1

Step 2: Check L size n
        i.      if n<=1
                a)  Goto Step 4

Step 3: If first<last
        i.      Call func(L,fisrt,last)
        ii.     Mid=(first+last)/2
        iii.    Call Sort(L,first,mid)
        iv.     Call Sort(L,mid+1,last)
        v.      Call merge(L,first,mid,mid+1,last)

Step 4: Exit
```

Sub-Algorithm: func (L,first,last)

Step 1: Initialise
- i. m=0
- ii. k=first

Step 2: Repeat while k<=last
- i. If L[k]>L[k+1]
  - a) temp=L[k]
  - b) L[k]=L[k+1]
  - c) L[k+1]=temp
- ii. k=k+2

Step 3: Initialise k=first+1
- i. Repeat while k<=last
  - a) A[m]=L[k]
  - b) m=m+1
  - c) k=k+2

Step 4: Initialise x=first and k=first
- i. Repeat while k<=last
  - a) L[x]=L[k]
  - b) x=x+1
  - c) k=k+2

Step 5: Initialise k=0
- i. Repeat while k<m
  - a) L[x]=A[k]
  - b) x=x+1
  - c) k=k+1

Sub-Algorithm: Merge (L, first, mid, mid+1, last)

Step 1: Initialise i=first, j=mid+1 and k=0

Step 2: Repeat while i<=mid and j<=last
- i. If L[i]<L[j]
  - a) temp[k++]=L[i++]
- ii. Else
  - a) temp[k++]=L[j++]

Step 3: Repeat while i<=mid
- i. temp[k++]=L[i++]

Step 4: Repeat while j<=last
- i. temp[k++]=L[j++]

Step 5: Initialise i=first and j=0
- i. Repeat while i<=last
  - a) L[i]=temp[j]
  - b) i=i+1
  - c) j=j+1

4. Pictorial representation

The Fig. 1 presents the pictorial representation of the proposed algorithm.

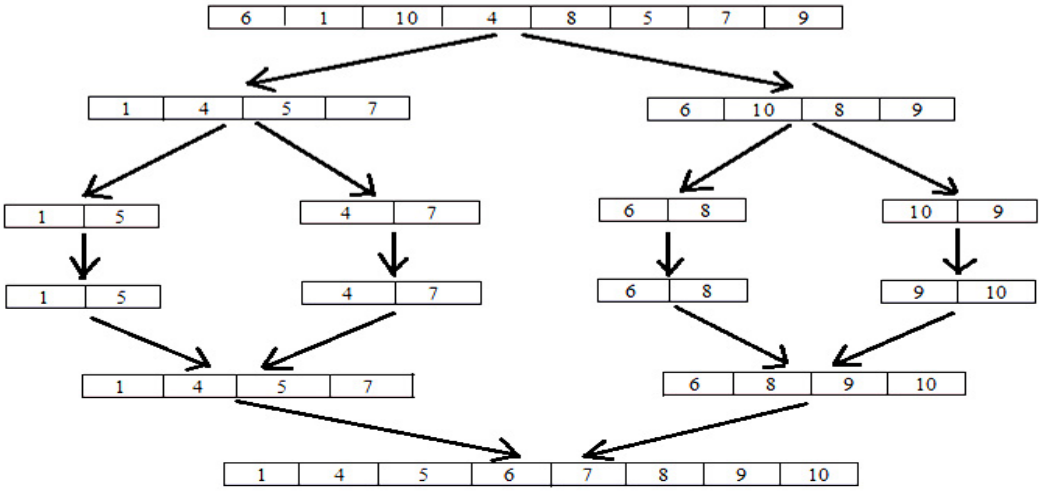

Fig. 1. Pictorial representation of Modified Merge Sort.

5. Time complexity

While calculating the time complexity of the newly proposed modified merge sort, following things are considered:-

- If the input size, n is small enough such that n<=c where c= constant. Then T(n)=Θ(1)
- Dividing L into two sub arrays requires Θ(1) time
- Applying the modified function requires Θ(n) time
- Solving the two equal or nearly equal sub arrays requires 2T(n/2) time
- Merging the two sorted arrays requires Θ(n) time

Hence, the recurrence for the modified merge sort is given by:-

$T(n) = c$ for n=1

$= 2T(n/2) + c_1 n + c_2 n$ otherwise

Where c_1 and c_2 are constants and $c_1 + c_2 = c$ (approx.)

Thus, the time complexity T(n)= O(n log n) as shown in Fig. 2.

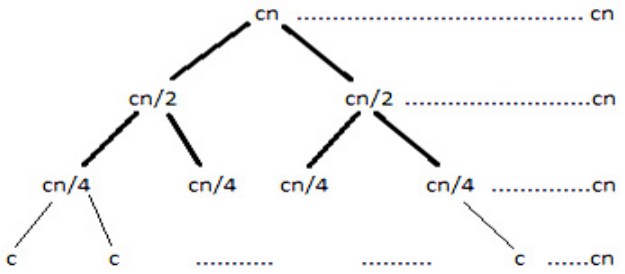

Fig. 2. Recursive tree for the proposed algorithm.

$$T(n) = cn \log n + cn$$
$$= O(n \log n)$$

6. Comparative execution analysis

The proposed algorithm is a combination of Max Min sorting algorithm[9] and Merge Sort algorithm yet it follows Divide and Conquer strategy. It has got a time complexity of $O(n \log_2 n)$. It is efficient compared to some conventional sorting algorithms and removes the major drawback of the Max Min Sorting algorithm[9]. The table [Table 1] below shows the distinguishing features of the proposed algorithm compared to other sorting algorithms.

Table 1. Comparative study of different Sorting algorithm.

Features	Max Min Sorting Algorithm[9]	Enhanced Merge Sort	Merge Sort	Quick Sort
Method	Selection	Selection + Merging	Merging	Partition
Sorting approach	Iterative	Recursive	Recursive	Recursive
Speed and Efficiency	Slow	Faster than Merge Sort and Quick Sort	Faster	Faster
Best case time complexity	$O(n^2)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$
Average case time complexity	$O(n^2)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$
Worst case time complexity	$O(n^2)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n^2)$
Space Complexity	O (1)	O (n)	O(n)	$O(\log_2 n)$

The execution times of the proposed algorithm have been calculated against different values of n (no. of array elements). These execution times are compared with two most efficient conventional Divide and Conquer Sorting algorithms i.e. Merge Sort and Quick Sort and have been plotted in a graph as shown in Fig. 3.

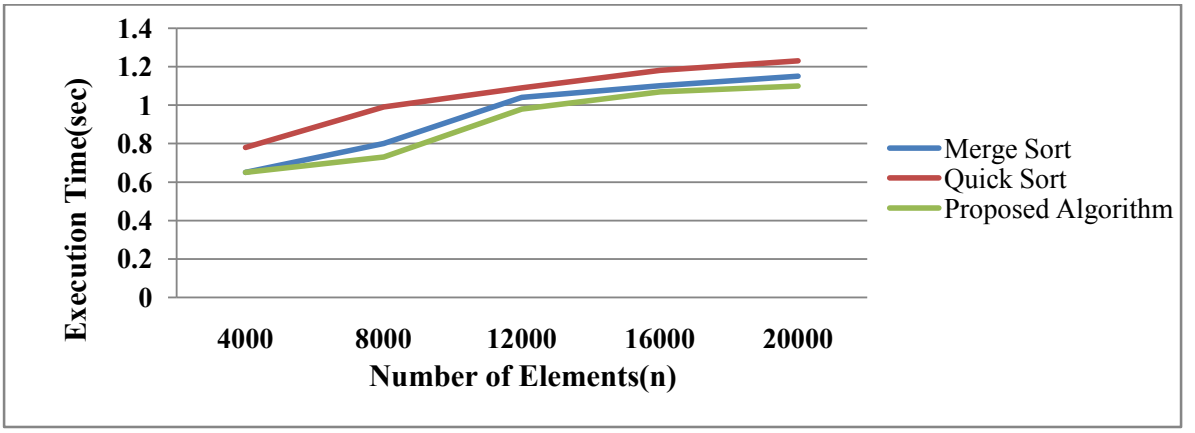

Fig. 3. Comparative execution times of various Sorting algorithms.

From the above graph, it is seen that when n=4000, both Merge Sort and the proposed algorithm takes same time to execute. As n increases,

Execution time of Merge Sort > Execution time of the proposed algorithm

On the other hand, the execution time of Quick Sort is much larger than the other two Sorting algorithms.

The proposed algorithm also reduces the number of steps as is required by the Max Min Sorting algorithm[9]. Thus, it removes the drawback and executes faster than the fastest Merge Sort algorithm.

7. Conclusion

Both the conventional and proposed algorithms have a time complexity of O(n log n). But the basic difference between them is that the former method executes faster if the array size is quite large while the new algorithm sort the array by simultaneously dividing the array into odd-even position sub-lists. Before doing the division, it performs a pair-wise sorting of the array. As a result, the array becomes partially sorted. In this way, it takes less time to sort a small as well as a large list. Since Merge Sort has a wide application in various file storage and databases, a modified version of the technique puts fewer burdens to the programmer to have a quick access of an item.

8. Future scope

Although the proposed algorithm sorts a large array in about negligible time compared to other Divide and Conquer algorithms yet it could not remove the major drawback of the conventional Merge Sort algorithm. It still requires a stack space of O(n). We shall try to overcome such problem in our future works.

References

1. A. V. Aho, J. E. Hopcroft, J. D. Ullman, "*Data Structures & Algorithms*", 2nd Edition, ISBN13: 9780201000238, Chapter 8, Page: 366-425.
2. G.S. Baluja, " *Data Structure through C*", Fourth Edition, ISBN13: 9788174462855, Chapter 10, Page 541-550.
3. S. Lipschutz , "*Data structure & Algorithm*", Schaum's Outlines Tata McGraw Hill 2nd Edition , ISBN13: 9780070991309.
4. Adamson, Iain T. ,"*Data structures and algorithms: A First Course*", ISBN13: 9783540760474, Page: 79-85.
5. B. R. Bhowmik, "*Design and Analysis of Algorithms*", Second Edition, ISBN 978-93-5014-135-9, Page: 14-92.
6. Knuth, D."The Art of Computer programming Sorting and Searching", 2nd edition, vol.3. Addison- Wesley, 1998.
7. Flores, I. "Analysis of Internal Computer Sorting". J.ACM 7,4 (Oct. 1960), 389- 409.
8. S. Paira, S. Chandra, S. S. Alam, P. S. Dey, "*A Review Report on Divide and Conquer Sorting Algorithms*", National Conference on Electrical, Electronics, and Computer Engineering, ISBN: 978-93-833-0383-0, November 7-8, 2014, IEEE CALCON 2014.
9. S. Paira, S. Chandra, S. S. Alam, S. S. Patra, "*Max Min Sorting Algorithm---A New Approach of sorting*", International Journal of Technological Exploration and Learning (IJTEL), ISSN: 2319-2135, April 2014, Vol. 3, No. 2, pp. 405-408.
10. S. Paira, A. Agarwal, Sk. S. Alam, S. Chandra, "*Doubly Inserted Sort: A Partially Insertion Based Dual Scanned Sorting Algorithm"*, Emerging Research in Computing, Information, Communication and Applications (ERCICA), DOI: 10.1007/978-81-322-2550-8_2, July 31-August 01, 2015, pp. 11-20, Springer India 2015.
11. ccis2k.org/iajit/PDF/vol.7,no.1/9.pdf
12. E. Horowitz and A. Zorat, "Divide-and-Conquer for Parallel Processing", *IEEE Transactions on Computers*, vol. C-32, no. 6, pp.582-585, June 1983.
13. J. L. Wolf, Daniel M. Dias, and P. S. Yu, "A Parallel Sort Merge Join Algorithm for Managing Data Skew", *IEEE Transactions on Parallel and Distributed Systems*, vol. 4,no. 1,pp. 70-86, Jan. 1993.
14. www.ijarcsse.com/docs/papers/Volume_3/3_March2013/V3I3-0319.pdf